How to Build a Real-Time Currency Converter with JavaScript

Building a currency converter is one of those projects that looks simple on the surface but teaches you a surprising amount about working with APIs, handling asynchronous data, and building responsive UIs. Whether you are adding multi-currency support to an e-commerce checkout or building a standalone tool, this tutorial walks you through every step.

By the end, you will have a fully working currency converter built with vanilla JavaScript -- no frameworks, no build tools, just HTML, CSS, and the Fetch API.

Prerequisites

Before you start, make sure you have:

- Basic knowledge of HTML, CSS, and JavaScript
- A modern browser (Chrome, Firefox, Edge, Safari)
- A free API key from AllRatesToday for live exchange rate data
- A text editor (VS Code, Sublime Text, or anything you prefer)

AllRatesToday provides real-time exchange rates for 160+ currencies, updated every 60 seconds, with a free tier that includes 300 requests per month. That is more than enough for development and light production use.

Step 1: Setting Up the HTML Structure

Start with a clean HTML file that includes a form with two currency selectors, an amount input, and a result display area.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Currency Converter</title>
```

```
  <style>
    * { box-sizing: border-box; margin: 0; padding: 0; }
    body { font-family: system-ui, sans-serif; background: #f8fafc; display: flex; justify-
content: center; align-items: center; min-height: 100vh; }
    .converter { background: white; border-radius: 12px; padding: 2rem; box-shadow: 0 4px 24px rgb
a(0,0,0,0.08); width: 100%; max-width: 420px; }
    .converter h1 { font-size: 1.5rem; margin-bottom: 1.5rem; color: #111827; }
    .field { margin-bottom: 1rem; }
    .field label { display: block; font-size: 0.875rem; font-weight: 600; color: #374151; margin-
bottom: 0.25rem; }
    .field input, .field select { width: 100%; padding: 0.75rem; border: 1px solid #d1d5db; border
-radius: 8px; font-size: 1rem; }
    .swap-btn { display: block; margin: 0.5rem auto; background: none; border: 1px solid #d1d5db;
border-radius: 50%; width: 40px; height: 40px; cursor: pointer; font-size: 1.2rem; }
    .result { background: #f0fdf4; border-radius: 8px; padding: 1rem; margin-top: 1rem; text-
align: center; font-size: 1.25rem; font-weight: 600; color: #166534; }
    .error { background: #fef2f2; color: #991b1b; }
    #convert-btn { width: 100%; padding: 0.75rem; background: #2ed06e; color: white; border:
none; border-radius: 8px; font-size: 1rem; font-weight: 600; cursor: pointer; margin-top:
0.5rem; }
    #convert-btn:hover { background: #25a557; }
    #convert-btn:disabled { background: #9ca3af; cursor: not-allowed; }
  </style>
</head>
<body>
  <div class="converter">
    <h1>Currency Converter</h1>
    <div class="field">
      <label for="amount">Amount</label>
      <input type="number" id="amount" value="1" min="0" step="any">
    </div>
    <div class="field">
      <label for="from-currency">From</label>
      <select id="from-currency"></select>
    </div>
    <button class="swap-btn" id="swap-btn" title="Swap currencies">&#8645;</button>
    <div class="field">
      <label for="to-currency">To</label>
      <select id="to-currency"></select>
    </div>
    <button id="convert-btn">Convert</button>
    <div class="result" id="result" style="display:none;"></div>
  </div>
  <script src="converter.js"></script>
</body>
</html>
```

This gives you a clean, responsive card layout. The two `<select>` elements will be populated dynamically with currency codes from the API.

Step 2: Fetching Exchange Rates from the API

Now create `converter.js`. The first thing you need is a function that fetches live exchange rates. AllRatesToday's API follows a straightforward REST pattern.

```
const API_KEY = 'YOUR_API_KEY'; // Replace with your AllRatesToday API key
const BASE_URL = 'https://api.allratestoday.com/v1';

async function fetchRates(baseCurrency) {
  const response = await fetch(
    `${BASE_URL}/latest?apikey=${API_KEY}&base=${baseCurrency}`
  );

  if (!response.ok) {
    throw new Error(`API error: ${response.status} ${response.statusText}`);
  }

  const data = await response.json();
  return data.rates;
}
```

This function takes a base currency code (like `USD`), calls the API, and returns an object of exchange rates. The `rates` object looks something like `{ EUR: 0.92, GBP: 0.79, JPY: 149.50, ... }`.

Populating the Currency Dropdowns

You also need a list of available currencies. You can fetch this from the API or hard-code the most common ones. Here is a dynamic approach:

```
async function populateCurrencies() {
  const rates = await fetchRates('USD');
  const currencies = ['USD', ...Object.keys(rates)];

  const fromSelect = document.getElementById('from-currency');
  const toSelect = document.getElementById('to-currency');

  currencies.forEach(code => {
    fromSelect.add(new Option(code, code));
    toSelect.add(new Option(code, code));
  });

  fromSelect.value = 'USD';
  toSelect.value = 'EUR';
}
```

Step 3: Building the Conversion Logic

The conversion itself is straightforward arithmetic. If you have the rates relative to a base currency, converting any amount from one currency to another is a single division and multiplication.

```
function convert(amount, fromRate, toRate) {
  if (fromRate <= 0 || toRate <= 0) {
    throw new Error('Invalid exchange rate');
  }
  return (amount / fromRate) * toRate;
}
```

Wire up the convert button:

```
let cachedRates = {};
let cacheTimestamp = 0;
const CACHE_TTL = 5 * 60 * 1000; // 5 minutes

async function getRates(base) {
  const now = Date.now();
  if (cachedRates[base] && (now - cacheTimestamp) < CACHE_TTL) {
    return cachedRates[base];
  }

  const rates = await fetchRates(base);
  cachedRates[base] = rates;
  cacheTimestamp = now;
  return rates;
}

async function handleConvert() {
  const btn = document.getElementById('convert-btn');
  const resultDiv = document.getElementById('result');
  const amount = parseFloat(document.getElementById('amount').value);
  const from = document.getElementById('from-currency').value;
  const to = document.getElementById('to-currency').value;

  if (isNaN(amount) || amount <= 0) {
    resultDiv.textContent = 'Please enter a valid amount';
    resultDiv.className = 'result error';
    resultDiv.style.display = 'block';
    return;
  }

  btn.disabled = true;
  btn.textContent = 'Converting...';

  try {
```

```
    const rates = await getRates(from);
    const result = amount * rates[to];
    resultDiv.textContent = `${amount.toLocaleString()} ${from} = ${result.toLocaleString(undefine
d, { minimumFractionDigits: 2, maximumFractionDigits: 4 })} ${to}`;
    resultDiv.className = 'result';
  } catch (error) {
    resultDiv.textContent = `Error: ${error.message}`;
    resultDiv.className = 'result error';
  } finally {
    resultDiv.style.display = 'block';
    btn.disabled = false;
    btn.textContent = 'Convert';
  }
}
```

Step 4: Adding Error Handling and Caching

Production-grade code needs to handle network failures, invalid responses, and API rate limits gracefully. Here is an improved fetch function with retry logic:

```
async function fetchWithRetry(url, retries = 3, delay = 1000) {
  for (let i = 0; i < retries; i++) {
    try {
      const response = await fetch(url);

      if (response.status === 429) {
        // Rate limited -- wait and retry
        const waitTime = delay * Math.pow(2, i);
        console.warn(`Rate limited. Retrying in ${waitTime}ms...`);
        await new Promise(resolve => setTimeout(resolve, waitTime));
        continue;
      }

      if (!response.ok) {
        throw new Error(`HTTP ${response.status}: ${response.statusText}`);
      }

      return await response.json();
    } catch (error) {
      if (i === retries - 1) throw error;
      await new Promise(resolve => setTimeout(resolve, delay));
    }
  }
}
```

Client-Side Caching Strategy

Exchange rates do not change every millisecond. Caching responses for a few minutes saves API calls and speeds up your app. The `getRates()` function above already implements a simple TTL cache. For more advanced scenarios, consider storing rates in `localStorage` so they survive page refreshes:

```
function saveToLocalStorage(key, data) {
  localStorage.setItem(key, JSON.stringify({
    data,
    timestamp: Date.now()
  }));
}

function loadFromLocalStorage(key, ttl) {
  const item = localStorage.getItem(key);
  if (!item) return null;

  const { data, timestamp } = JSON.parse(item);
  if (Date.now() - timestamp > ttl) {
    localStorage.removeItem(key);
    return null;
  }
  return data;
}
```

Step 5: Complete Working Example

Here is the full `converter.js` file putting everything together:

```
const API_KEY = 'YOUR_API_KEY';
const BASE_URL = 'https://api.allratestoday.com/v1';
const CACHE_TTL = 5 * 60 * 1000;

// --- Caching ---
function saveCache(base, rates) {
  localStorage.setItem(`rates_${base}`, JSON.stringify({ rates, ts: Date.now() }));
}

function loadCache(base) {
  const raw = localStorage.getItem(`rates_${base}`);
  if (!raw) return null;
  const { rates, ts } = JSON.parse(raw);
  return (Date.now() - ts < CACHE_TTL) ? rates : null;
}

// --- API ---
```

```
async function fetchRates(base) {
  const cached = loadCache(base);
  if (cached) return cached;

  const res = await fetch(`${BASE_URL}/latest?apikey=${API_KEY}&base=${base}`);
  if (res.status === 429) throw new Error('Rate limit exceeded. Try again shortly.');
  if (!res.ok) throw new Error(`API error: ${res.status}`);

  const data = await res.json();
  saveCache(base, data.rates);
  return data.rates;
}

// --- UI ---
async function populateCurrencies() {
  try {
    const rates = await fetchRates('USD');
    const currencies = ['USD', ...Object.keys(rates).sort()];
    const fromEl = document.getElementById('from-currency');
    const toEl = document.getElementById('to-currency');

    currencies.forEach(c => {
      fromEl.add(new Option(c, c));
      toEl.add(new Option(c, c));
    });
    fromEl.value = 'USD';
    toEl.value = 'EUR';
  } catch (err) {
    console.error('Failed to load currencies:', err);
  }
}

async function handleConvert() {
  const btn = document.getElementById('convert-btn');
  const resultDiv = document.getElementById('result');
  const amount = parseFloat(document.getElementById('amount').value);
  const from = document.getElementById('from-currency').value;
  const to = document.getElementById('to-currency').value;

  if (isNaN(amount) || amount <= 0) {
    showResult('Enter a valid amount greater than zero.', true);
    return;
  }

  btn.disabled = true;
  btn.textContent = 'Converting...';

  try {
    const rates = await fetchRates(from);
    const converted = amount * rates[to];
    showResult(
      `${amount.toLocaleString()} ${from} = ${converted.toLocaleString(undefined, { minimumFractio
nDigits: 2, maximumFractionDigits: 4 })} ${to}`
```

```
    );
  } catch (error) {
    showResult(error.message, true);
  } finally {
    btn.disabled = false;
    btn.textContent = 'Convert';
  }
}

function showResult(message, isError = false) {
  const el = document.getElementById('result');
  el.textContent = message;
  el.className = isError ? 'result error' : 'result';
  el.style.display = 'block';
}

// --- Event listeners ---
document.getElementById('convert-btn').addEventListener('click', handleConvert);
document.getElementById('swap-btn').addEventListener('click', () => {
  const fromEl = document.getElementById('from-currency');
  const toEl = document.getElementById('to-currency');
  [fromEl.value, toEl.value] = [toEl.value, fromEl.value];
});
document.getElementById('amount').addEventListener('keydown', (e) => {
  if (e.key === 'Enter') handleConvert();
});

// Initialize
populateCurrencies();
```

Save both files in the same directory, replace `YOUR_API_KEY` with your actual key from AllRatesToday, and open `index.html` in a browser. You should see a working converter that fetches live rates and caches them locally.

Performance Tips

Once your converter is working, here are some ways to make it faster and more reliable:

- **Cache aggressively.** Exchange rates update every 60 seconds at most. A 5-minute client-side cache eliminates redundant requests and keeps you well within AllRatesToday's free tier of 300 requests per month.
- **Prefetch common pairs.** If your users mostly convert between USD, EUR, and GBP, fetch those rates on page load so conversions feel instant.
- **Use the `base` parameter wisely.** Instead of fetching all rates twice (once for each direction), fetch rates for the "from" currency and multiply directly.

- **Show stale data while refreshing.** Display the cached result immediately and update it silently when fresh data arrives. Users prefer a fast stale answer over a slow accurate one.
- **Debounce input.** If you add live-as-you-type conversion, debounce the input handler so you are not firing API calls on every keystroke.

FAQ

What API should I use for a JavaScript currency converter?

AllRatesToday provides a free exchange rate API with real-time rates updated every 60 seconds for 160+ currencies. It includes an official JavaScript/Node.js SDK available on npm, making integration straightforward.

Can I build a currency converter with vanilla JavaScript?

Yes. Using the Fetch API and any exchange rate REST API, you can build a fully functional currency converter with plain HTML, CSS, and JavaScript. No frameworks are required, as this tutorial demonstrates.

How do I handle API rate limits in a currency converter?

Cache exchange rates locally and only fetch new rates when needed. AllRatesToday's free tier includes 300 requests per month, which is more than sufficient when combined with the client-side caching strategy shown above.

Wrapping Up

You now have a complete, production-ready currency converter built with vanilla JavaScript. The key takeaways are: use a reliable API like AllRatesToday for accurate real-time rates, implement client-side caching to minimize API calls, and always handle errors gracefully so your users never see a broken UI.

If you want to take this further, consider adding historical rate charts, supporting cryptocurrency conversions, or integrating the AllRatesToday npm package for a cleaner server-side integration.

Get your free API key and start building.